Simulated annealing and genetic algorithms

CE 377K

Stephen D. Boyles

Spring 2015

1 Introduction

Different optimization problems must be solved in different ways. Even just for the simple examples developed at the start of this course, the way that one solves the shortest path problem, for instance, is very different from the way that one solves the transit frequency setting problem. Later in this course, we will learn examples of these types of specific solution methods. But first, we start with a broad class of solution methods which can be applied to nearly any optimization problem.

In optimization, there is often a tradeoff between how widely applicable a solution method is, and how efficient or effective it is at solving specific problems. For any one specific problem, a tailor-made solution process is likely much faster than a generally-applicable method, but generating such a method requires more effort and specialized knowledge, and is less "rewarding" in the sense that the method can only be more narrowly applied.

So, the first solution methods you will see in this course are very widely applicable. They are examples of *heuristics*, which are not guaranteed to find the global optimum solution, but tend to work reasonably well in practice. In practice, they only tend to be applied for very large or very complicated problems which cannot be solved exactly in a reasonable amount of time with our current knowledge. But until you learn the more specialized solution methods later in the class, you can also use these even for smaller or simpler optimization problems too.

Many engineers are initially uncomfortable with the idea of a heuristic. After all, the goal of an optimization problem is to find an optimal solution, so why should we settle for something which is only approximately "optimal," often without any guarantees of how approximate the solution is? First, for very complicated problems a good heuristic can often return a reasonably good solution in much less time than it would take to find the exact, global optimal solution. For many practical problems, the cost improvement from a reasonably good solution to an exactly optimal one is not worth the extra expense (both time and computational hardware) needed, particularly if the heuristic gets you within the margin of error based on the input data.

Heuristics are also very, very common in psychology and nature. If I give someone a map and ask them to find the shortest-distance route between two points in a city by hand, they will almost certainly not formulate an mathematical model and solve it to provable optimality. Instead, they use mental heuristics (rules of thumb based on experience) and can find paths which are actually quite good. Many of the heuristics are inspired by things seen in nature.

An example is how ant colonies find food. When a lone wandering ant encounters a food source, it returns to the colony and lays down a chemical pheremone. Other ants who stumble across this pheremone begin to follow it to the food source, and lay down more pheremones, and so forth. Over time, more and more ants will travel to the food source, taking it back to the colony, until it is exhausted at which point the pheremones will evaporate. Is this method the most optimal way to gather food? Perhaps not, but it performs well enough for ants to have survived for millions of

years!

Another example is the process of evolution through natural selection. The human body, and many other organisms, function remarkably well in their habitats, even if their biology is not exactly "optimal."[1] One of the most common heuristics in use today, and one described later in these notes, is based on applying principles of natural selection and mutation to a "population" of candidate solutions to an optimization problem, using an evolutionary process to identify better and better solutions over time.

Of course, we should not content ourselves with a heuristic solution when it is possible to efficiently find an exact one, and much of the rest of the course will involve identifying specific types of problems which can in fact be solved exactly. But to get us started, these notes describe two heuristics: *simulated annealing*, and *genetic algorithms*, both of which can be applied to many different optimization problems.

2 Simulated Annealing

Simulated annealing is a simple heuristic that makes an analogy to metallurgy, but this analogy is best understood after the heuristic is described.

As a better starting point for understanding simulated annealing, consider the following "local search" heuristic. (For now, descriptions will be a bit vague; more precise definitions will follow soon.) Local search proceeds as follows:

1. Choose some initial feasible solution $\mathbf{x} \in X$, and calculate the value of the objective function $f(\mathbf{x})$.
2. Generate a new feasible solution $\mathbf{x}' \in X$ which is close to the current solution $\mathbf{x}$.
3. Calculate $f(\mathbf{x}')$
4. If $f(\mathbf{x}') \leq f(\mathbf{x})$, the new solution is better than the old solution. So update the current solution by setting $\mathbf{x}$ equal to $\mathbf{x}'$.
5. Otherwise, the old solution is better, so leave $\mathbf{x}$ unchanged.
6. Return to step 2 and repeat until we are unable to make further progress.

One way to visualize local search is a hiker trying to find the lowest elevation point in a mountain range. In this analogy, the park boundaries are the feasible set, the elevation of any point is the objective function, and the location of the hiker is the decision variable. In local search, starting from his or her initial position, the hiker looks around and finds a nearby point which is lower than their current point. If such a point can be found, they move in that direction and repeat the process. If every neighboring point is higher, then they stop and conclude that they have found the lowest point in the park.

[1] For instance, in humans the retina is "backwards," creating a blind spot; in giraffes, the laryngeal nerve takes an exceptionally long and roundabout path; and the the descent of the testes makes men more vulnerable to hernias later in life.

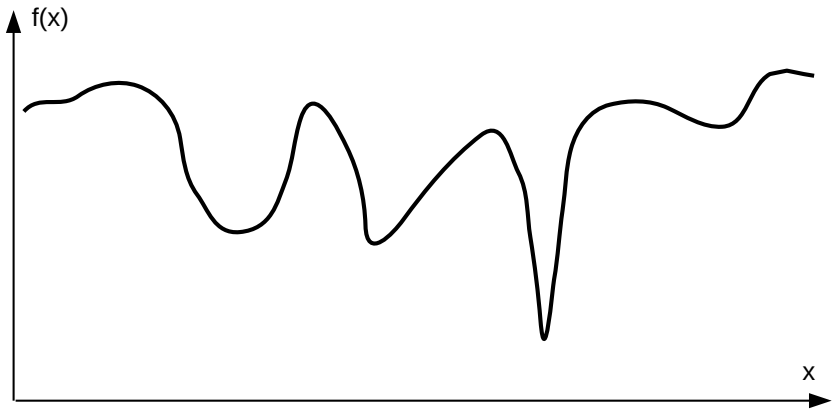

Figure 1: Moving downhill from the current location may not lead to the global optimum.

It is not hard to see why this strategy can fail; if there are multiple local optima, the hiker can easily get stuck in a point which is not the lowest (Figure 1). Simulated annealing attempts to overcome this deficiency of local search by allowing a provision for occasionally moving in an uphill direction, in hopes of finding an even lower point later on. Of course, we don't always want to move in an uphill direction and have a preference for downhill directions, but this preference cannot be absolute if there is any hope of escaping local minima.

Simulated annealing accomplishes this by making the decision to move or not probabilistic, introducing a *temperature* parameter T which controls how likely you are to accept an "uphill" move. When the temperature is high, the probability of moving uphill is large, but when the temperature is low, the probability of moving uphill becomes small. In simulated annealing, the temperature is controlled with a *cooling schedule*. Initially, the temperature is kept high to encourage a broad exploration of the feasible region. As the temperature decreases slowly, the solution is drawn more and more to lower areas. The cooling schedule can be defined by an initial temperature T_0, a final temperature T_f, the number of search iterations n at a given temperature level, and a scaling factor $k \in (0, 1)$ which is applied every n iterations to reduce the temperature.

Finally, because the search moves both uphill and downhill, there is no guarantee that the final point of the search is the best point found so far. So, it is worthwhile to keep track of the best solution $\mathbf{x}^*$ encountered during the algorithm. (This is analogous to the hiker keeping a record of the lowest point observed, and returning to that point when done searching.)

So, the simulated annealing algorithm can be stated as follows:

1. Choose some initial feasible solution $\mathbf{x} \in X$, and calculate the value of the objective function $f(\mathbf{x})$.
2. Initialize the best solution to the initial one $\mathbf{x}^* \leftarrow \mathbf{x}$.
3. Set the temperature to the initial temperature: $T \leftarrow T_0$
4. Repeat the following steps n times:
   (a) Generate a new feasible solution $\mathbf{x}'$ which neighbors $\mathbf{x}$.
   (b) If $f(\mathbf{x}') < f(\mathbf{x}^*)$, it is the best solution found so far, so update $\mathbf{x}^* \leftarrow \mathbf{x}'$.

(c) If $f(\mathbf{x}') \leq f(\mathbf{x})$, it is a better solution than the current sone, so update $\mathbf{x} \leftarrow \mathbf{x}'$.

(d) Otherwise, update $\mathbf{x} \leftarrow \mathbf{x}'$ with probability $\exp(-[f(\mathbf{x}') - f(\mathbf{x})]/T)$

5. If $T > T_f$, then reduce the temperature ($T \leftarrow kT$) and return to step 4.

6. Report the best solution found $\mathbf{x}^*$

The key step is step 4d. Notice how the probability of "moving uphill' depends on two factors: the temperature, and how much the objective function will increase. The algorithm is more likely to accept an uphill move if it is only slightly uphill, or if the temperature is high. The exponential function captures these effects while keeping the probability between 0 and 1. A few points require explanation.

How should the cooling schedule be chosen? Unfortunately, it is hard to give general guidance here. Heuristics often have to be "tuned" for a particular problem: some problems do better with higher temperatures and slower cooling (k values closer to 1, n larger), others work fine with faster cooling. When you use simulated annealing, you should try different variations of the cooling schedule to identify one that works well for your specific problem.

How should an initial solution be chosen? It is often helpful if the initial solution is relatively close to the optimal solution. For instance, if the optimization problem concerns business operations, the current operational plan can be used as the initial solution for further optimization. However, it's easy to go overboard with this. You don't want to spent so long coming up with a good initial solution that it would have been faster to simply run simulated annealing for longer starting from a worse solution. The ideal is to think of a good, quick rule of thumb for generating a reasonable initial solution; failing that, you can always choose the initial solution randomly.

How do I define a neighboring solution for step 4a? Again, this is problem-specific, and one of the decisions that must be made when applying simulated annealing. A good neighborhood definition should involve points which are "close" to the current solution in some way, but ensure that feasible solutions are connected in the sense that any two feasible solutions can be reached by a chain of neighboring solutions. For the examples in the previous set of notes, some possibilities (emphasis on *possibilities*, there are other choices) are:

Transit frequency setting problem: The decision variables $\mathbf{n}$ are the number of buses assigned to each route. Given a current solution $\mathbf{n}$, a neighboring solution is one where exactly one bus has been assigned to a different route.

Scheduling maintenance: The decision variables are $\mathbf{x}$, indicating where and when maintenance is performed, and $\mathbf{c}$, indicating the condition of the facilities. In this problem, the constraints completely define $\mathbf{c}$ in terms of $\mathbf{x}$, so for simulated annealing it is enough to simply choose a feasible solution $\mathbf{x}$, then calculate $\mathbf{c}$ using the state evolution equations. In this problem, there is no reason to perform less maintenance than the budget allows each year. If the initial solution is chosen in this way, then a neighborhing solution might be one where one maintenance activity on a facility is reassigned to another facility that same year. If the resulting solution is infeasible (because the resulting $\mathbf{c}$ values fall below the minimum threshold), then another solution should be generated.

Facility location problem: The decision variables are the intersections that the three terminals are located at, L_1, L_2, and L_3. Given current values for these, in a neighboring solution two of the three terminals are at the same location, but one of the three has been assigned to a different location.

Shortest path problem: The decision variables specify a path between the origin and destination. Given a current path, a neighboring path is one which differs in only intersection. (Can you think of a way to express this mathematically?)

How do I perform a step with a given probability? Most programming languages have a way to generate a uniform random variable between 0 and 1. If we want to perform a step with probability p, generate a random number from the continuous uniform $(0, 1)$ distribution. If this number is less than p, perform the step; otherwise, do not.

2.1 Example with facility location

This section demonstrates simulated annealing, on an instance of the facility location problem from the previous set of notes. In this problem instance, the road grid consists of ten north-south and ten east-west streets. The cost of locating a terminal at each of the locations is shown in Figure 2 (these numbers were randomly generated). There are 30 customers, randomly located throughout the grid, as shown in Figure 3. In these figures, the coordinate system (x, y) is used to represent points, where x represents the number of blocks to the right of the upper-left corner, and y represents the number of blocks below the upper-left corner.

The cooling schedule is as follows: the initial temperature is $T_0 = 1000$, the final temperature is $T_f = 100$, $k = 0.75$, and the temperature is reduced every 8 iterations. (These values were chosen through trial-and-error, and work reasonably well for this problem.) The first several iterations are shown below.

The algorithm begins with an initial solution, generated randomly. Suppose this solution locates the three facilities at coordintaes (5,1), (8,6), and (6,1), respectively; the total cost associated with this solution is 158.0 cost units. A neighboring solution is generated by picking one of the facilities (randomly) and assigning it to another location (randomly). Suppose that the third facility is reassigned to location (1,0), so the new candidate solution locates the facilities at (5,1), (8,6), and (1,0). This solution has a cost of 136.4 units, which is lower than the current solution. So, simulated annealing replaces the current solution with the new one.

In the next iteration, suppose that the first facility is randomly chosen to be reassigned to (8,2), so the candidate solution is (8,2), (8,6), and (1,0). This solution has a cost of 149.1, which is higher than the current cost of 136.4. The algorithm will not always move to such a solution, but will only do so with probability calculated as in Step 4c:

$$p = \exp(-[f(\mathbf{x}') - f(\mathbf{x})]/T) = \exp(-[149.1 - 136.4]/100) = 0.881$$

This probability is high, because (being one of the early iterations) the temperature is set high. If the temperature were lower, the probability of accepting this move would be lower as well.

Supposing that the move is accepted, the algorithm replaces the current solution with the candidate and continues as before. If, on the other hand, the move is rejected, the algorithm generates another

8.7	13.5	11.0	10.2	6.3	6.4	12.4	13.5	9.7	6.8
5.8	12.4	8.8	15.1	14.7	13.9	9.0	5.2	8.7	12.7
11.8	9.8	12.4	12.6	6.3	13.3	7.7	10.2	13.6	13.5
5.4	6.9	11.1	8.4	14.2	10.9	10.7	11.7	8.1	8.9
4.9	12.5	10.8	6.6	12.0	6.8	11.9	9.2	9.5	10.0
14.4	9.7	8.6	11.6	8.0	6.7	12.7	5.9	7.6	14.1
7.7	9.6	6.4	9.9	9.2	13.3	12.3	14.7	15.0	5.1
7.6	14.7	7.1	5.5	5.5	9.7	9.7	14.4	7.9	15.1
10.9	12.5	9.2	12.1	14.8	6.4	6.1	10.5	12.5	10.8
12.0	5.9	13.1	10.0	11.9	10.0	8.6	8.4	10.7	5.3

Figure 2: Cost of locating facilities at each intersection.

	x	x					x		
	x			x			x		
		x		xx					
xx				x	x				
					x	x		x	
xx	x			x	x				
							xx		
	xx						xx		
			x					xx	

Figure 3: Locations of customers.

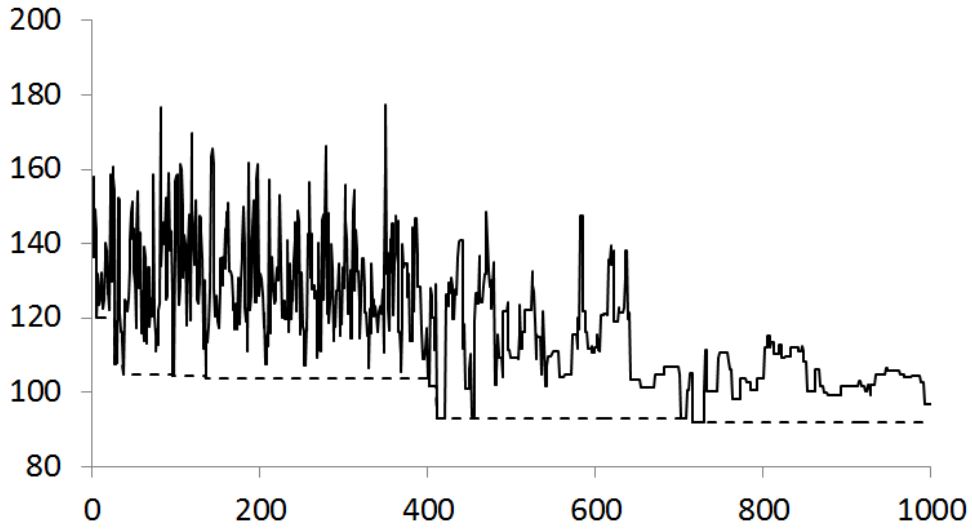

Figure 4: Progress of simulated annealing. Solid line shows current solution cost, dashed line best cost so far.

candidate solution based on the same current solution (5,1), (8,6), and (1,0). The algorithm continues in the same way, reducing the temperature by 25% every 8 iterations.

The progress of the algorithm until termination is shown in Figure 4. The solid line shows the cost of the current solution, while the dashed line tracks the cost of the best solution found so far. A few observations are worth making. First, in the early iterations the cost is highly variable, but towards termination the cost becomes more stable. This is due to the reduction in temperature which occurs over successive iterations. When the temperature is high, nearly any move will be accepted so one expects large fluctuations in the cost. When the temperature is low, the algorithm is less likely to accept cost-increasing moves, so fewer fluctuations are seen. Also notice that the best solution was found shortly after iteration 700. The final solution is not the best, although it is close.

The best facility locations found by simulated annealing are shown in Figure 5, with a total cost of 92.1.

3 Genetic Algorithms

Another heuristic is the genetic algorithm. In contrast to simulated annealing, where there is a single search happening through the feasible region (recall the analogy with the hiker), genetic algorithms maintain a larger *population* of solutions which reflect a diverse set of points within the feasible region. Genetic algorithms work by attempting to improve the population from one iteration to the next.

The process of improvement is intended to mimic the processes of natural selection, reproduction, and mutation which are observed in biology. For the sake of our purposes, *natural selection* means identifying solutions in the population which have good (low) values of the objective function.

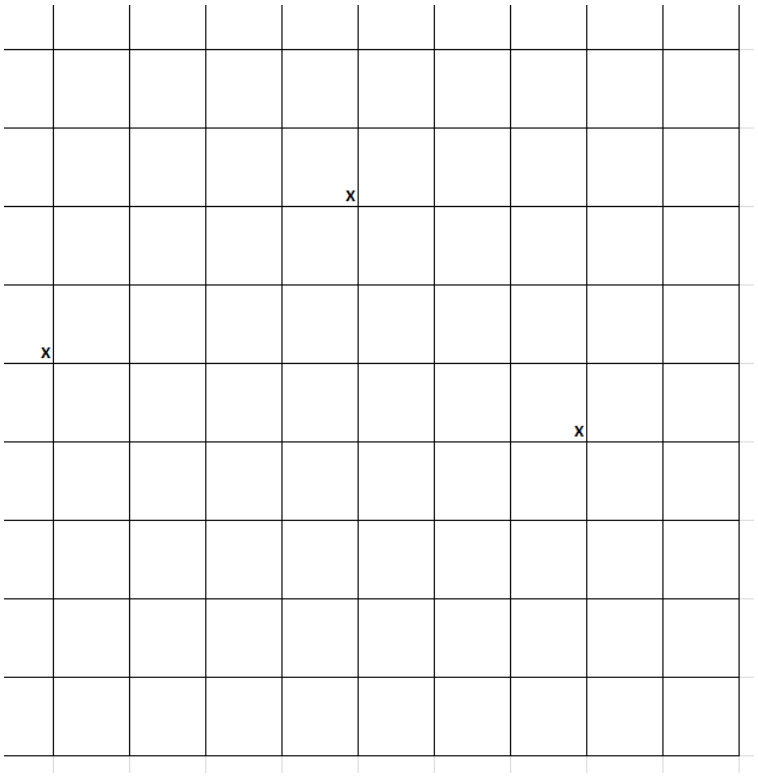

Figure 5: Locations of facilities found by simulated annealing (cost 92.1).

The hope is that there are certain aspects or patterns in these solutions which make them good, which can be maintained in future generations. Given an initial population as the first generation, subsequent generations are created by choosing good solutions from the previous generation, and "breeding" them with each other in a process called *crossover* which mimics sexual reproduction: new "child" solutions are created by mixing characteristics from two "parents." Lastly, there is a random *mutation* element, where solutions are changed externally with a small probability. If all goes well, after multiple generations the population will tend towards better and better solutions to the optimization problem.

At a high level, the algorithm is thus very straightforward and presented below. However, at a lower level, there are a number of careful choices which need to be made about how to implement each step for a particular problem. Thus, each of the steps is described in more detail below. The high-level version of genetic algorithms is as follows:

1. Generate an initial population of N feasible solutions (generation 0), set generation counter $g \leftarrow 0$.
2. Create generation $g + 1$ in the following way, repeating each step N times:
   (a) Choose two "parent" solutions from generation g.
   (b) Combine the two parent solutions to create a new solution.
   (c) With probability p, mutate the new solution.
3. Increase g by 1 and return to step 2 unless done.

Although not listed explicitly, it is a good idea to keep track at every stage of the best solution $\mathbf{x}^*$ found so far. Just like in simulated annealing, there is no guarantee that the best solution will be found in the last generation. So, whenever a new solution is formed, calculate its objective function value, and record the solution if it is better than any found so far.

Here are explanations of each step in more detail. It is very important to realize that *there are many ways to do each of these steps* and that what is presented below is one specific example intended to give the general flavor while sparing unnecessary complications at this stage. There is much more to genetic algorithms than just a few pages of notes!

Generate an initial population of feasible solutions: In contrast with simulated annealing, where we had to generate an initial feasible solution, with genetic algorithms we must generate a larger population of initial feasible solutions. While it is still desirable to have these initial feasible solutions be reasonably good, it is also very important to have some diversity in the population. Genetic algorithms work by combining characteristics of different solutions together. If there is little diversity in the initial population the difference between successive generations will be small and progress will be slow.

Selection of parent solutions: Parent solutions should be chosen in a way that better solutions (lower objective function values) are more likely to be chosen. This is intended to mimic natural selection, where organisms better adapted for a particular enviroment are more likely to reproduce. One way to do this is through *tournament selection*, where a number of solutions from the old generation are selected randomly, and the one with the best objective function value is chosen as the first parent. Repeating the "tournament" again, randomly select another subset of solutions from the old generation, and choose the best one as the second parent. The number of entrants in the tournament is a parameter that you must choose.

Combining parent solutions: This is perhaps the trickiest part of genetic algorithms: how can we combine two feasible solutions to generate a new feasible solution which retains aspects of both parents? The exact process will differ from problem to problem. Here are some ideas, based on the example problems from the previous set of notes:

Transit frequency setting problem: The decision variables are the number of buses on each route. Another way of "encoding" this decision is to make a list assigning each bus in the fleet to a corresponding route. (Clearly, given such a list, we can construct the n_r values by counting how many times a route appears.) Then, to generate a new list from two parent lists, we can assign each bus to either its route in one parent, or its route in the other parent, choosing randomly for each bus.

Scheduling maintenance: As described above, optimal solutions in this problem must always exhaust the budget each year. So, along the lines of the transit frequency setting problem, make a list of the number of maintenance actions which will be performed in each year, noting the facility assigned to each maintenance action in the budget. To create a new solution, for each maintenance action in the budget choose one of the facilities assigned to this action from the two parents, selecting randomly. Once this list is obtained, it is straightforward to calculate $\mathbf{x}$ and $\mathbf{c}$.

Facility location problem: For each facility in the child solution, make its location the same as the location of that facility in one of the two parents (chosen randomly).

Mutating solutions: Mutation is intended to provide additional diversity to the populations, avoiding stagnation and local optimal solutions. Mutation can be accomplished in the same way that neighbors are generated in simulated annealing. The probability of mutation should not be too high — as in nature, most mutations are harmful — but enough to escape from local optima. The mutation probability p can be selected by trial and error, determining what works well for your problem.

3.1 Example with facility location

This section demonstrates genetic algorithms on the same facility location problem used to demonstrate simulated annealing. The genetic algorithm is implemented with a population size of 100, over ten generations. Tournaments of size 3 are used to identify parent solutions, and the mutation probability is 0.05. The initial population is generated by locating all the facilities are completely at random.

To form the next generation, 100 new solutions need to be created, each based on combining two solutions from the previous generation. These two parents are chosen using the tournament selection rule, as shown in Figure 6. The two winners of the tournament locate the three facilities at (2,3), (1,7), (8,7); and (1,4), (7,6), (5,0) respectively. These combine in the following way:

1. The first facility is located either at (2,3) or (1,4); randomly choose one of them, say, (2,3).
2. The second facility is located either at (1,7) or (7,6); randomly choose one of them, say, (1,7).
3. The third facility is located either at (8,7) or (5,0); randomly choose one of them, say, (5,0).

This gives a new solution (2,3), (1,7), (5,0) in the next generation, as shown in Figure 6. With 5% probability, this solution will be "mutated." If this solution is selected for mutation, one of the three facility is randomly reassigned to another location. For instance, the facility at (1,7) may be reassigned to (0,4). This process is repeated until all 100 solutions in the next generation have been created.

Figure 7 shows the progress of the algorithm over ten generations. The solid line shows the average cost in each generation. The dashed line shows the cost of the best solution found so far, and the crosses show the cost of each of the solutions comprising the generations. Since lower-cost alternatives are more likely to win the tournaments and be selected as parents, the average cost of each generation decreases. Figure 8 shows the locations of the terminals in the best solution found.

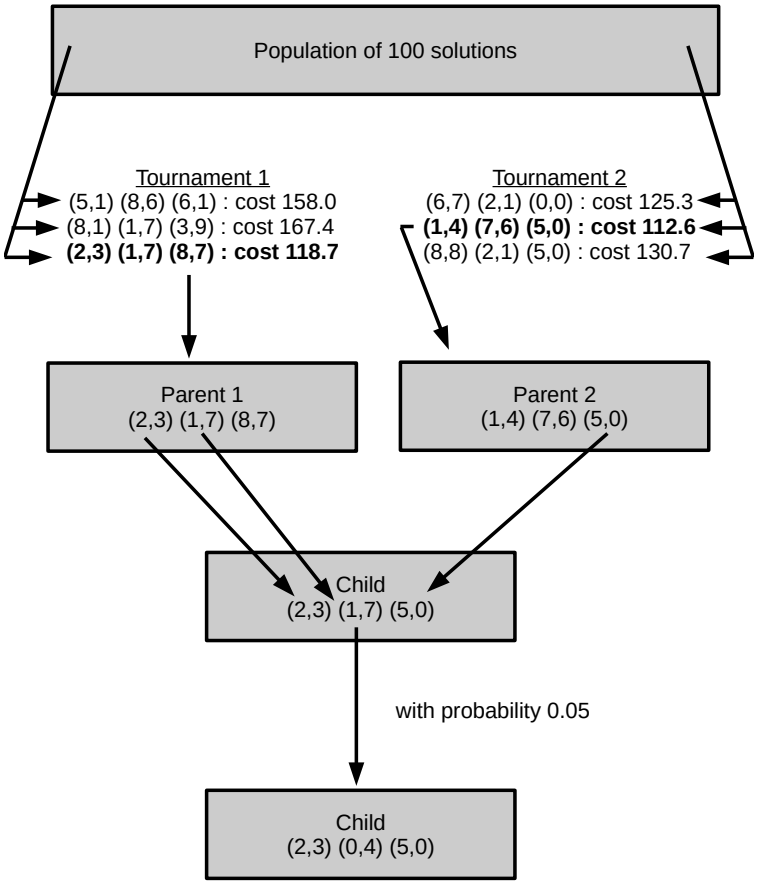

Figure 6: Generation of a new solution: reproduction and mutation.

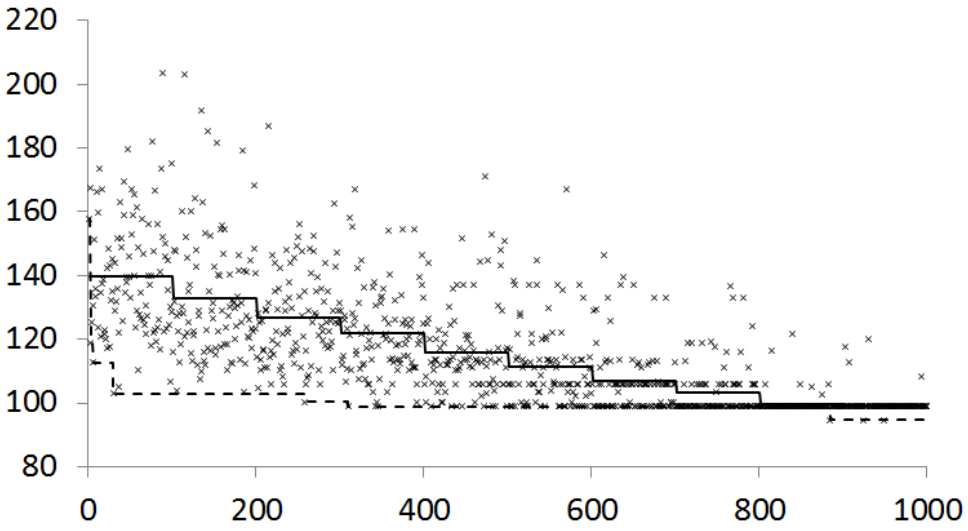

Figure 7: Progress of genetic algorithm. Solid line shows average generation cost, dashed line best cost so far, crosses cost of individual solutions.

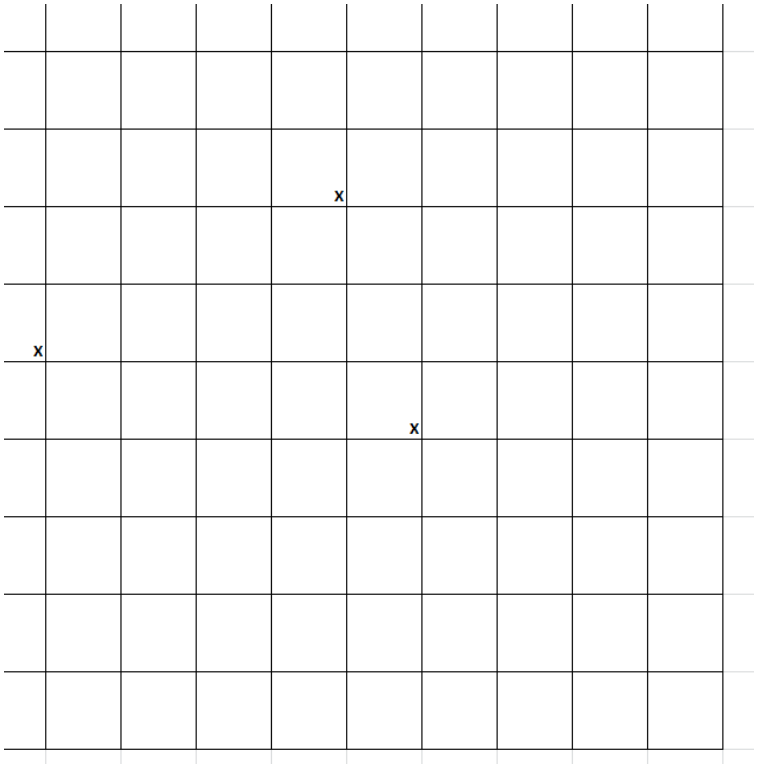

Figure 8: Locations of facilities found by genetic algorithm (cost 94.7).